A Trajectory-Preserving Synchronization Method for Collaborative Visualization

Lewis W.F. Li, Frederick W.B. Li, and Rynson W.H. Lau

Abstract—In the past decade, a lot of research work has been conducted to support collaborative visualization among remote users over the networks, allowing them to visualize and manipulate shared data for problem solving. There are many applications of collaborative visualization, such as oceanography, meteorology and medical science. To facilitate user interaction, a critical system requirement for collaborative visualization is to ensure that remote users will perceive a synchronized view of the shared data. Failing this requirement, the user's ability in performing the desirable collaborative tasks will be affected. In this paper, we propose a synchronization method to support collaborative visualization. It considers how interaction with dynamic objects is perceived by application participants under the existence of network latency, and remedies the motion trajectory of the dynamic objects. It also handles the false positive and false negative collision detection problems. The new method is particularly well designed for handling content changes due to unpredictable user interventions or object collisions. We demonstrate the effectiveness of our method through a number of experiments.

Index Terms—Collaborative visualization, network latency, motion synchronization, distributed synchronization.

1 Introduction

Collaborative visualization [10] allows geographically separated users to access a shared virtual environment to visualize and manipulate datasets for problem solving without physical travel. Example works include fluid dynamics visualization [27], volume visualization [1] and medical data visualization [26]. In contrast to those working individually with standalone visualization applications, research studies have found that users working in groups through collaborative visualization applications can often work out a better solution for a given problem [17]. To facilitate collaborative dataset manipulation for visualization, CSpray [21] was designed to comprise a "spray-paint can metaphor" for users to edit a dataset in a graphical way. A control mechanism is provided for users to modify the dataset in a mutually exclusive manner. If a user changes the dataset, updates of the dataset will be broadcasted to remote users. However, the system does not address the inconsistency problem of dynamic objects due to network latency. Hence, if we conduct visualization on a time-dependent dataset [9], such as thunderstorms and tornados, synchronization of the dataset among remote users would be difficult. First, as this type of dataset changes continuously over time, it is difficult to guarantee that each of these changes will be reported timely to the remote users throughout the collaboration session. Second, as different users may be connected to each other or to the server via different network routes, they may perceive different amounts of network latency, and hence delay, in receiving the update messages. Although it is possible to introduce a further delay for the updated information to be presented to the users at a synchronized moment [7], it will substantially affect the interactivity of the collaboration.

Recently, we have developed a method to support global-wise synchronization for collaborative applications [14]. The method runs a reference simulator for each dynamic object on the application server. Each of the clients interested in the object, including those that access the objects as well as the owner of the object, will execute a gradual synchronization process on the local copy of the object to align its motion to that of the reference simulator running at the server. Our results show that the method effectively reduces the network latency by half and quickly align the motion of the replicated object to that of the original dynamic object. However, the method still suffers from a high error during the period when an interaction has just occurred and before the interaction message has reached the client, causing the users to make inappropriate decisions. It may also leads to the false positive and false negative collision problems as discussed later.

In this paper, we present a trajectory-preserving synchronization method, which significantly extends our previous work [14] to support collaborative visualization. It considers how spatial changes and interactions of dynamic objects are affected by network latency. A set of procedures have been developed to correct the motion trajectory of the dynamic objects. In addition, solutions have also been provided to handle the false positive and false negative collision detection problems. To demonstrate the effectiveness of our method, we have conducted experiments on a prototype system for flow visualization [25]. With this prototype, users may manipulate dynamic objects with the CyberGloves, which are electronic gloves for sensing hand and finger motions, to intervene the flow of a dataset for visualization. The dynamic and interactive nature of this prototype provides an efficacious testbed for verifying the effectiveness of the new method.

The rest of the paper is organized as follows. Section 2 briefly summarizes related work. Section 3 outlines the foundation of our method. Section 4 presents in detail our trajectory-preserving synchronization method. Section 5 shows how the new method handles object collisions. Section 6 studies the performance of the proposed method with a number of experiments. Finally, Section 7 briefly concludes the work presented in this paper.

2 Related Work

2.1 Collaborative Applications

A unique characteristic of collaborative applications is the need to distribute state updates to remote sites over the network to update the states of the shared objects at these sites. Because of network latency, different remote sites may receive the updates after different amounts of delay, causing the view discrepancy problem at these sites. Nevertheless, traditional applications such as [2] and [12] may still work well under the existence of network latency as long as the state updates are received by the remote sites in a correct order. This is

- *Lewis W.F. Li is with Department of Computer Science at City University of Hong Kong, Hong Kong, E-Mail: kwfli@cs.cityu.edu.hk.*
- *Frederick W.B. Li and Rynson W.H. Lau are with Department of Computer Science at University of Durham, United Kingdom., E-Mail: {Frederick.Li | Rynson.Lau} @durham.ac.uk.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

because these applications typically have a large time gap between any two consecutive updates as compared with the network latency. As such, the network latency becomes insignificant and the users can implicitly perceive synchronized application content at all times.

However, collaborative applications that involve time-dependent data and continuous user interaction may be different from the above applications. The state update events in these applications are *continuous* [18] in nature. In order for users to interact with the system based on the same updated view of the data, the updates need to be presented to the remote users either without any delay or at least within a very short period of time. However, this is not trivial to achieve. An early attempt to explore the discrepant views among remote users on shared data due to network latency was done in DEVA3 [22], which claimed that inconsistency may be tolerated if latency is small enough. Some works have been conducted to study how the delay in delivering state updates will affect the interactivity of collaborative applications [4, 21]. To cope with the latency problem, adaptations could be performed at either the user or the system side. For user side adaptation, [24] proposes to explicitly disclose delay information to the users and let the users adjust their behavior by themselves to cope with the state discrepancy problem. Unfortunately, this arrangement is subjective to different users and would heavily slow down the user interaction.

For system side adaptation, a popular approach is to use dead reckoning [16]. With this approach, the controlling client of a dynamic object runs a motion predictor for the object. Other clients accessing the object also run the same motion predictor to drive the motion of the local copies of the object. The controlling client is required to keep track of the error between the actual and predicted motions of the object, and sends the updated motion information to the other clients when the error is higher than a given threshold. Although this approach is very simple, it does not guarantee that the state of a shared object could be synchronized among all the remote clients.

In [18], a local-lag mechanism is proposed to address this problem. When the controlling client issues a state update of the dynamic object, the update will be sent to the remote clients immediately but not to the sender itself until a local-lag period is expired. This is to reduce the discrepancy between the sender and the receivers. However, as different pairs of clients may suffer from different amounts of latency, a single value of local-lag can only be used to synchronize two clients, the sender and the receiver, but not among a number of clients. In [3], users make use of the reference state information from the server to correct the states of their local copies of the dynamic objects. Again, there is no control mechanism to guarantee that the states generated at a client would be synchronized with those at other clients.

2.2 Clock Synchronization

The synchronization problem has also been studied by researchers working on clock synchronization. In particular, Network Time Protocol (NTP) [19] has been adopted as a standard for computers connected via the Internet to synchronize their clocks to within 10ms of error. It relies on selecting and filtering time information from a set of time servers. On the other hand, there are also different strategies proposed in adjusting the local clock when the correct time information is received [10, 12, 20]. *Backward correction* [11] and *forward correction* [23] are two approaches, which make a backward or a forward adjustment on the clock value, respectively. Undesirably, they introduce a time discontinuity problem to the clock. [13] addresses this problem by speeding up or slowing down a clock to synchronize it against a reference clock. However, it incurs a severe run-time overhead as it needs to adjust the time once at every clock tick. Hence, it generally requires hardware support. To reduce the overhead, [15] proposes an adaptive method for clock synchronization. It uses a time server to propagate time information to the clients via a re-synchronization process for the clients to determine their clock drift rates. The time between two consecutive re-synchronization processes will be shortened or lengthened based on the drift rate of a client clock. It is set inversely proportion to clock drift rate. At the client, clock correction is performed by extrapolating the clock continuously with the newest clock drift rate determined in the latest re-synchronization process.

Although methods used in clock synchronization appear to address the synchronization problem in collaborative visualization, it is difficult to apply them directly to address the problem. First, the time value in clock synchronization is only a parameter with a single degree-of-freedom, while the motion parameters of the datasets for visualization may have three or higher degrees-of-freedom. Second, the clock information is periodic, i.e., the occurrence of a time event is predictable. This simplifies the synchronization problem. In contrast, the motion of the datasets for visualization is likely unpredictable, especially when user interactions or object collisions are possible. Third, in clock synchronization, the time server is the prime reference for all clocks, which only need to synchronize to the changes from the time server. In collaborative visualization, however, any user may initiate its own actions to manipulate a dataset and such actions need to be synchronized among all the users.

3 Foundation

3.1 Consistency Control Model

In [14], we proposed a *relaxed consistency control model* to synchronize the object states among remote clients in collaborative applications. We observed that application users would likely pay more attention on the state trajectory of a dynamic object, i.e., the continuous sequence of state changes of the object, rather than on the individual states of the object in order for them to determine their actions to respond. Hence, we proposed to relax the strict time-dependent consistency control requirement on individual states of a replicated object among all relevant clients to allow the state trajectories of the replicated object among the relevant sites to deviate from that of the correct one by an acceptable amount. Formally speaking, given that the states of a replicated object at two remote sites at time t are $s_i(t)$ and $s_j(t)$, the state discrepancy D of the object between the two sites during any time period T_a and T_b should be smaller than an application specific tolerance, ξ. Therefore,

$$D = \int_{T_a}^{T_b} | s_i(t) - s_j(t) | \, dt < \xi \tag{1}$$

This relaxed model could be reverted back to the original strict time-dependent consistency control model if we shorten the time period so that $T_a = T_b$ and set tolerance $\xi = 0$.

3.2 Gradual Synchronization

To implement the consistency control model, we have developed a *gradual synchronization method* [14] to trade accuracy of individual states of a dynamic object for the preservation of the state trajectory of the object. We assume that a collaborative application has a client-server architecture. The server runs a simulator, called a *reference simulator*, for each dynamic object. This reference simulator serves as a standard reference for synchronizing the motion of all copies of the object at different remote clients. Each client interested in an object will also run a simulator as a local copy of the object. This method effectively reduces the latency of a client to obtain the updated state of an object from a double round-trip time delay to a single one.

To simulate object motions, we need to apply appropriate motion equations [8] to drive the motion of the objects. For example, we may apply a first-order predictor (or a more advanced method [5]) to drive an object when it is under the user's control:

$$p_{new} = p + t \times V \tag{2}$$

where p is the current position of the dynamic object, t is the time difference between p and p_{new}, and V is the motion vector of the

object. When we need to simulate object interactions and responses, we may apply a second-order predictor instead:

$$p_{new} = p + Vt + At^2/2 \quad (3)$$

$$V_{new} = V + At \quad (4)$$

where A is the acceleration vector of the dynamic object, and t is the time difference between V and V_{new}. For the simulation of flows, such as water, smoke or fire, we may apply appropriate motion equations according to their behaviors [6].

During run-time, two motion timers T_s and T_c are maintained at the server and the client, respectively. They are the virtual clocks indicating how long a dynamic object has been performing certain movement as perceived by the server and by the client. Hence, they represent t in Eq. (2) to (4). When a dynamic object changes its motion, each interested client will gradually align the motion of its local copy of the object to that of the reference simulator at the server by adjusting the increment rate of T_c of the object. The simulator of the object at the client and the reference simulator of the object at the server are said to be synchronized when $T_c = T_s$.

In general, this method successfully maintains the consistency of dynamic objects in collaborative applications, except between the period when an interaction has just occurred and before the update message reaches the remote client. During this period, the two sites can have a very high discrepancy. Although this discrepancy will be settled shortly after the update message is received at the remote client, it still produces a high visual error during this period. This can be serious if the interactions occur frequently.

4 Trajectory-Preserving Synchronization

The new synchronization method extends our earlier method [14] by considering how interactions are perceived by the remote users or the server in the existence of network latency. It includes separate mechanisms for handling *client-server* and *client-client synchronizations*. To simplify our discussion, Figures 1 to 3 help illustrate our method graphically. We assume that client **A** initiates a motion change to a dynamic object, which can be a user controlled object or a primitive object of a dataset in flow simulation. A motion command is then generated as a result of the motion change and sent to the server to update the motion of the corresponding reference simulator **S**. Concurrently, client **B** is visualizing the change of the object and needs to gather updates of the object from the reference simulator running at the server.

4.1 Client-Server Synchronization

As shown in Figure 1, before a new motion occurs at P_{init}, the motions of the dynamic object at client **A** and of its reference simulator **S** are synchronized. When the object is driven to move in a new direction V_{new} by a script, by a user interaction or by an object collision, a motion command is generated. However, as it will take time for this motion command to reach the server, **S** will continue to move in its current direction until P_{start}, when the server receives the motion command. This discrepant motion is represented as the expected motion vector V_r, and its motion would last for the time duration of a half round-trip delay.

To remedy the state discrepancy problem during motion change, we propose to adjust the motion of the dynamic object at **A** gradually to align with that of **S**. This motion remediation method helps minimize the state discrepancy raised during a motion change while preserving the motion trajectory as much as possible. It is shown by the blue arrows in Figure 2. First, instead of driving the dynamic object at **A** solely with V_{new}, we let it move in the direction of the vector sum of V_{new} and V_r for the time duration T^{est}_{A-S}, where T^{est}_{A-S} is the estimated latency between client **A** and the server collected from recent statistics [14]. Note that T^{est}_{A-S} is used instead of the real latency (half round-trip time) T_{A-S}, since the most updated T_{A-S} is not currently available at the client. After that, we set the motion of the dynamic object to move in the direction of V_{new}. Once the server has received the new motion command from **A**, it will reply the client by sending it the values of P_{start} and T_s. Upon the reception of such information, **A** will evaluate the residual state discrepancy V_{dis} as the vector difference between P_{curr} of the object at the server and the current position of the object at **A**, where $P_{curr} = P_{start} + (T_{A-S} + T_s) \times V_{new}$ and T_{A-S} is the most updated latency. With the value of V_{dis}, we may further remedy the motion of the dynamic object at **A** to move in the direction of the vector sum of V_{new} and V_{dis}. Hence, such motion could eventually be synchronized with that of **S** at P_{sync}. Finally, we again resume the motion of the object at **A** to move in the direction of V_{new}.

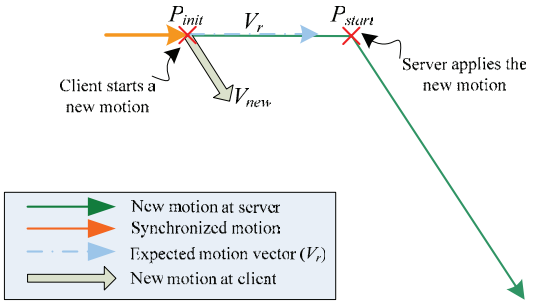

Fig. 1. The state discrepancy problem during motion change.

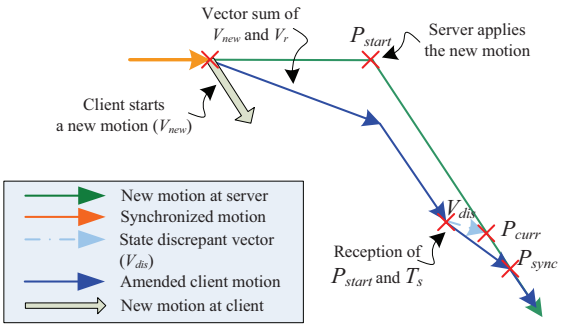

Fig. 2. Motion remediation in client-server synchronization.

4.2 Client-Client Synchronization

Figure 3 shows our motion remediation process to address the state discrepancy problem in client-client communications. Using the same scenario and notation as in the client-server synchronization example, we consider the situation that client **B** is interested in the motion of the dynamic object driven by client **A**. Hence, the server needs to propagate the motion information of the object to **B**. Before the object is driven to move in a new direction, the motions of **S** at the server and the local copy of the object at **B** are synchronized at P_{start}. Assuming that **S** is driven to move in a new direction V_{new} when an interaction occurs, V_{new} will then be propagated to **B**. Again, it will take time for this motion command to arrive at **B**. Thus, the local copy of the object at **B** is expected to continuous moving in its current direction until client **B** receives V_{new}. Then, **B** will estimate the state discrepancy V_{dis} as the vector difference between P_{curr} of the object at the server and the current position of it at **B**, where $P_{curr} = P_{start} + T^{est}_{S-B} \times V_{new}$. Note that T^{est}_{S-B} is the estimated latency between client **B** and the server collected from recent statistics. With V_{dis}, we may amend the motion of the object in **B** to move in the direction of the vector sum of V_{new} and V_{dis} such that it could eventually be closely synchronized with that of **S** at P_{sync}. At this point, we resume the motion of the local copy of the object to move in the direction of V_{new}. This adjusted client motion is shown as brown arrows in Figure 3. Hereafter, when client **B** receives the most updated latency T_{S-B}, it will adjust the increment rate of T_c using T_{S-B}, which is the gradual synchronization process for client **B** to remedy any residual state discrepancy [14].

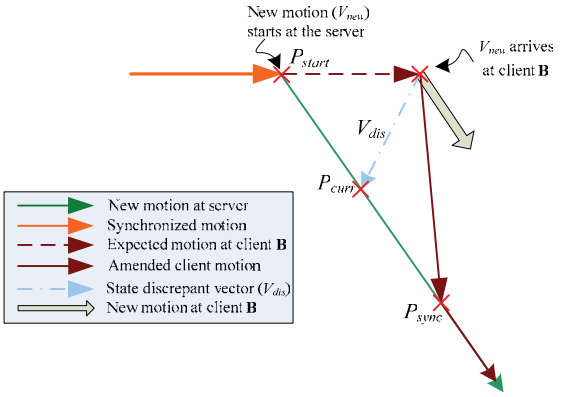

Fig. 3. Motion remediation in client-client synchronization.

4.3 Synchronization at an Arbitrary Moment

One important advantage of the new method is that a prior synchronized state between a dynamic object and its reference simulator is not required before a new synchronization cycle could be taken place. Hence, our method can synchronize new motion commands generated at any arbitrary moment, including during motion remediation. We explain this with the example shown in Figure 4. While client **A** is executing a motion remediation process, it sends out another motion command to indicate that the motion of the dynamic object has been changed to V_{new}. Although the previous object motion command at **A** has still not been synchronized with the reference simulator at the server, we may start a new client-server synchronization process for the new motion command by applying the new motion on the object immediately.

Instead of moving in V_{new}, the dynamic object at **A** moves in the direction of the vector sum of V_{new} and V_r for the time duration of T^{est}_{A-S}, followed by the direction of V_{new} until the client has received P_{start} and T_s from the server. Upon receiving such information, the client evaluates the residual state discrepancy V_{dis} as the vector difference between P_{curr} of the object at the server and the current position of the object at **A**, in the same way as described in Section 4.1. With V_{dis}, we may further amend the motion of the object at **A** to move in the direction of the vector sum of V_{new} and V_{dis} such that this motion and that of **S** could eventually be synchronized at P_{sync}. Finally, we resume the motion of the object at **A** to move in the direction of V_{new}.

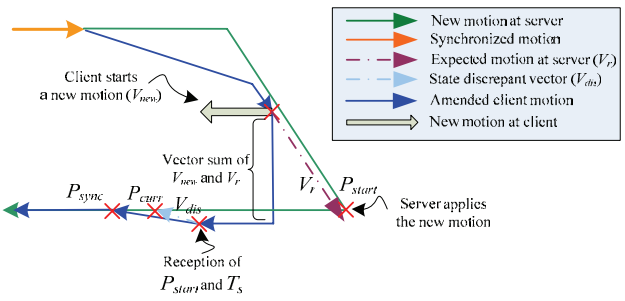

Fig. 4. Motion remediation at an arbitrary moment.

5 Handling Of Object Collisions

Another critical issue in collaborative visualization is to assert a consistent view among remote users in object collisions. This issue typically does not exist in a standalone visualization application, since updated states of dynamic objects are maintained and presented within a single client, where delay can be neglected. However, because of network latency and since each user may suffer from a different amount of network latency, object collision information may be presented to different users at different time moments. This may lead to inconsistent object collision results. This problem is still an open research topic.

To our knowledge, [20] is the only work that attempts to address this problem. It uses dead reckoning to guide the motions of dynamic objects and addresses the inconsistency problem as follows. First, when a client needs to detect possible collisions between two remote objects or between one remote object and one static object, it uses the local state information of these objects to compute a temporary collision result for visual presentation. The collision result may then be overridden by an updated one from the remote client when it is available. This may, however, introduce temporary inconsistency on the collision results among remote users. Second, when a client predicts that a collision will likely occur between its own controlled object and a remote object, the system would instruct the remote client to propagate the state of the remote object more frequently to this client. This helps decrease the error in evaluating the collision detection result at the client. This unfortunately would increase the network loading. In addition, the *false positive* and *false negative* collision results, which will be discussed next, are not considered in this method.

5.1 The Collision Problem

In our method, when an object collision occurs, we evaluate and interpret the collision response as motion commands according to the reactions of the colliding objects. The motion commands would then be fed as input to the motion predictors of the objects to update the object motions. In this way, our synchronization method could natively support the global-wise consistency of object collisions among the participants, i.e., the participating clients and the server. However, because of network latency, the new motion commands will still be received and interpreted by each participant at a different time moment. This may lead to a *false positive* or a *false negative* collision detection result, which corresponds to an invalid or a missing collision instance of an object, respectively.

We have identified all possible false positive and false negative collision detection results as shown in Table 1. We assume that the roles of the clients and of the server are the same as those mentioned in Section 4. From Table 1, there are mainly two reasons leading to the collision problem. First, when an object in client **A** changes its motion during motion remediation, i.e., same situation as Section 4.3, the motion of this object at client **A** and that of the reference simulator at the server will be different temporarily. Second, as it takes time for the motion commands to be delivered to client **B**, the motion of the object in **B** will also be different from that of the reference simulator at the server. In both cases, inconsistent collision results may be produced.

5.2 The Algorithm

To address the collision problem, we follow our reference simulator scheme, where the motion of a dynamic object maintained at a client only needs to be synchronized with that of the reference simulator at the server. Thus, if we could resolve the collision problem between each client (either client **A** or client **B**) with the server, we would then have handled the problem globally among all participants. Based on this observation, we convert the collision problem shown in Table 1 into two simpler problems, involving only two parties: client **A** – server (Table 2) and client **B** – server (Table 3). To address the collision problem, we may just handle individual cases shown in Tables 2 and 3 as follows:

- Cases (a), (d), (e) and (h): As the collision problem does not exist, no actions are required.
- Cases (c) and (g): A false positive collision occurs at the server. When a dynamic object changes its motion, the server will send an update message to client **B** to update the state of the object at **B**. To trade transient discrepancy of client **A** for global consistency, the collision event detected at the server will be sent

Table 1. Possible false positive and false negative collision problems (× indicates no collision and O indicates a collision)

Client A	Server	Client B	Collision Problem and Its Causes
×	×	×	No collisions occur. The collision problem does not exist.
×	×	O	Before the new motion command arrives at client **B**, the motion of the object at **B** leads to a collision. False positive collision occurs.
×	O	×	Client **A** issues a new motion command during motion remediation, which does not lead to a collision. However, such command causes a false positive collision when it reaches the server.
×	O	O	Client **A** issues a new motion command during motion remediation, which does not lead to a collision, while client **B** inherits the same state from the server. False positive collision occurs.
O	×	×	Client **A** issues a new motion command during motion remediation, which leads to a collision. However, this command does not cause a collision when it reaches the server, while client **B** inherits the same state from the server. False negative collision occurs.
O	×	O	Client **A** issues a new motion command during motion remediation, which leads to a collision. However, the new command does not cause a collision when it reaches the server. False negative collision occurs. On the other hand, before the new command arrives at client **B**, the motion of the object at **B** leads to a collision, which does not correspond to the one at client **A**. False positive collision occurs.
O	O	×	Client **B** does not receive the state update in time. False negative collision occurs.
O	O	O	Collision is detected correctly at all parties. No collision problems arise.

to both clients **A** and **B** to override the object motion there so that they will both have the same object state.

- Case (b): A false negative collision result occurs at the server. Since the server does not detect a collision, it will not send out a collision event, causing an inconsistency with client **A**. This problem only occurs when client **A** issues a new motion command during motion remediation, which leads to a collision. To trade transient discrepancy of client **A** for global consistency, we inhabit client **A** to perform collision detection until the motion remediation process has finished, which typically takes a round-trip time.
- Case (f): A false positive collision result occurs at client **B**. This happens only while the server is sending a motion command to client **B** but the original motion of the object at **B** has already led to a collision. This problem could not be avoided but could be quickly corrected by the new motion command from the server, and the inconsistent state could only last for a half round-trip time. As in case (b), to maintain global consistency, we inhabit client **B** to perform collision detection until the motion remediation process has finished. This situation would last for another half round-trip time.

Table 2. Client **A** and server collision problems

Case	Client A	Server
(a)	×	×
(b)	O	×
(c)	×	O
(d)	O	O

Table 3. Server and Client **B** collision problems

Case	Server	Client B
(e)	×	×
(f)	×	O
(g)	O	×
(h)	O	O

6 Results and Discussions

To study the performance of our method, we have developed a prototype to support flow visualization and dynamic user interaction. We have tested it on a set of PCs with a P4 2.0GHz CPU, 1GBytes RAM and a GeForce4 Ti4200 graphics card. Each client machine has a CyberGlove with a 3D tracker connected to it to capture the user's hand gesture and position. The CyberGlove allows a user to manipulate the objects in an intuitive way inside the prototype environment, where each unit of spatial distance is defined as 1m. The user may apply force to an object and use the object to intervene a flow simulation. In addition, we have connected the machines through TCP connections, which handle the packet lost problem automatically, and use timestamps to ensure message ordering. Figure 5 shows a series of screenshots from one of our experiments on flow visualization.

Before the experiments, we first collected the latency statistics of different network connections as show in Table 4. (Connection Overseas 1 measures the latency between Hong Kong and US, while Overseas 2 is created to model a long latency connection.)

Table 4. Latency information for different network connections

Category	Latency (ms)		
	Mean	Max.	Min
LAN (within a department)	5	7	0
Intranet (within a university)	40	57	32
Overseas 1 (modeling two nearby countries)	160	186	132
Overseas 2 (modeling two distant countries)	325	537	294

6.1 Experiment 1

This experiment compares the performance of the new method with our original synchronization method [14] and dead reckoning [16] in terms of accuracy. It measures the object position discrepancy experienced by relevant machines when two users are connected via an overseas link with a network latency of roughly 160ms as shown in Figure 6. In the experiment, we use first-order and second-order polynomials to model the motion of the object when it is being grasped to move and thrown out by a user, respectively.

To simplify the discrepancy measurement without loss of generality, we observe in this experiment the motion of a selected object, which is spherical in shape. The users have a full control on the motion of this object and use the object to intervene a flow simulation. However, to simplify this experiment, we have confined the motion of all the tiny particles of the flow so that they do not affect the motion of the selected object. During the experiment, a user (client **A**) is asked to pick up the selected object and move it around arbitrarily with a velocity of 1 m/s. After around 4 seconds, the user throws the object out with a velocity of 3m/s under 9.8m^2/s gravity, it hits the floor and bounces up and down several times. The elastic coefficient of the object is 0.7. Such interaction is observed by a remote user (client **B**). We have measured the position discrepancy of the object between client **A** – server (Figure 6(a)), server – client **B** (Figure 6(b)), and client **A** – client **B** (Figure 6(c)) during a fixed period of time. The position discrepancies exhibited by our original method, and the new method, and dead reckoning are shown in each of the three diagrams.

We assume that the positions of the object in the three machines are synchronized at 1s. As the user sudden changes the motion of the object, we can see that there is an increase in discrepancy in all three diagrams of Figure 6. In Figure 6(a), as dead reckoning does not perform any correction until when the server receives the update

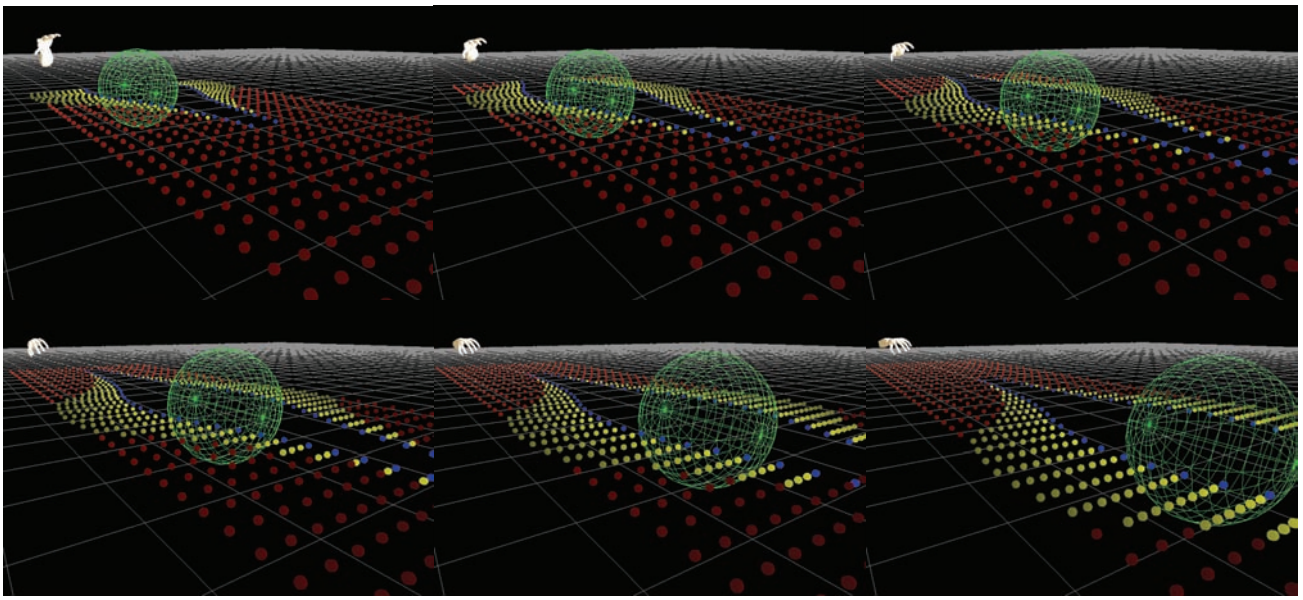

Fig. 5. Screen shots of our prototype for collaborative visualization.

message from client **A**, the discrepancy rises much more rapidly. In addition, it also lasts for longer as dead reckoning does not send out an update message to the server until the error between the actual and the predicted motions is high enough. With our previous method, when the object at client **A** changes motion, **A** would send out an update message to the server immediately and move the object at a reduced speed to minimize its discrepancy with the server. Hence, its discrepancy rises much slower than dead reckoning. The method that we propose here is even more aggressive. It modifies the motion of the object at client **A** to anticipate for the latency existed between **A** and the server. Hence, its discrepancy is further reduced.

When the server has received and applied the update to the object, the object discrepancy using dead reckoning drops immediately. This happens at around 1.2s. For our original and the new methods, we receive the update message at around 1.16s, after 160ms of latency. The reason for our methods to receive the update message slightly earlier than dead reckoning is that we do not need a thresholding process. Our original method would also have a sudden drop as we update the object position at the server to the actual position where the object started to change motion at client **A**. However, the discrepancy would not drop to zero as the object in client **A** has already moved some distance. This discrepancy will be further reduced as the object in client **A** continues to move in a reduced speed than that of the server until they are synchronized. With the new method, the discrepancy gradually reduces to zero (approximately) as we continue to correct the object motion at client **A** until it is synchronized with that in the server.

In Figure 6(b), the discrepancy of dead reckoning suddenly increases as the server suddenly correct the object location when it receives the update message from client **A**. This discrepancy continues to increase as the object in client **B** continues to move in its current direction until **B** receives the update message from the server. Then, the discrepancy suddenly drops to zero as **B** applies the update to the object. With our original method, the rise in discrepancy is similar to dead reckoning except that here it happens earlier at about 1.16s instead of 1.2s and drops earlier at about 1.32s. However, the discrepancy would not drop to zero as we move the object at client **B** to the location where the object started to change motion at the server. This discrepancy will be further reduced as the object in client **B** speeds up until it catches up with the object at the server. With the new method, the discrepancy gradually reduces to zero as we continue to correct the object motion at client **B**.

At about 4s, client **A** throws the object out, which hits the floor and bounces up and down several times. This leads to a series of object collisions. We discuss the performance of our method when we apply it to collision detection in the next experiment.

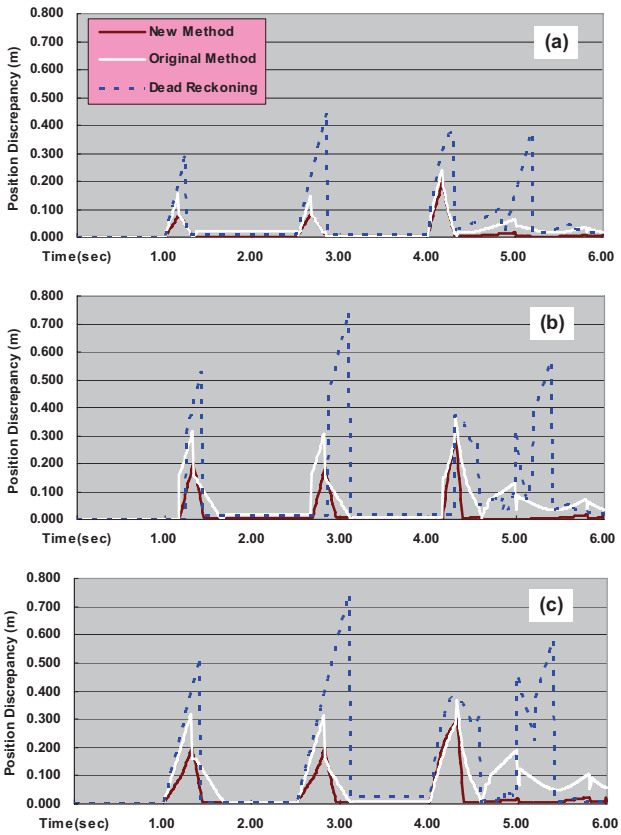

Fig. 6. Position discrepancies of the three methods, observed between: (a) client **A** and the server, (b) the server and client **B**, and (c) client **A** and client **B**.

In general, we may observe from Figure 6 that although the dead reckoning method is very simple, it produces high discrepancy during motion changes. Our original method is successful in reducing the discrepancy. However, through motion adjustment as

well as using the server as a reference, the new method not only further reduces the discrepancy but also shorten the duration of discrepancy, as observed in Figure 6(c).

6.2 Experiment 2

In this experiment, we follow similar settings as in Experiment 1 but focus more on the accuracy of the new method in handling object collisions under different types of network connection as shown in Table 4. The experiment is conducted by 4 users, clients **A** to **D**. Each of the users is connected to the server with a different network connection and takes turn to act as the *controller*, who throws the selected object out to initiate collisions, and the other three users would be the *observers*, who monitor the motion of the selected object. When throwing the object, the controller simply throws the object up to the sky and let it fall down on to the floor. Then, the object will bounce up and down a few times before it rests on the floor. During the experiment, we record the position discrepancy of the object between each of the users and the server. The results are shown in Figure 7, with each user taking turn to be the controller.

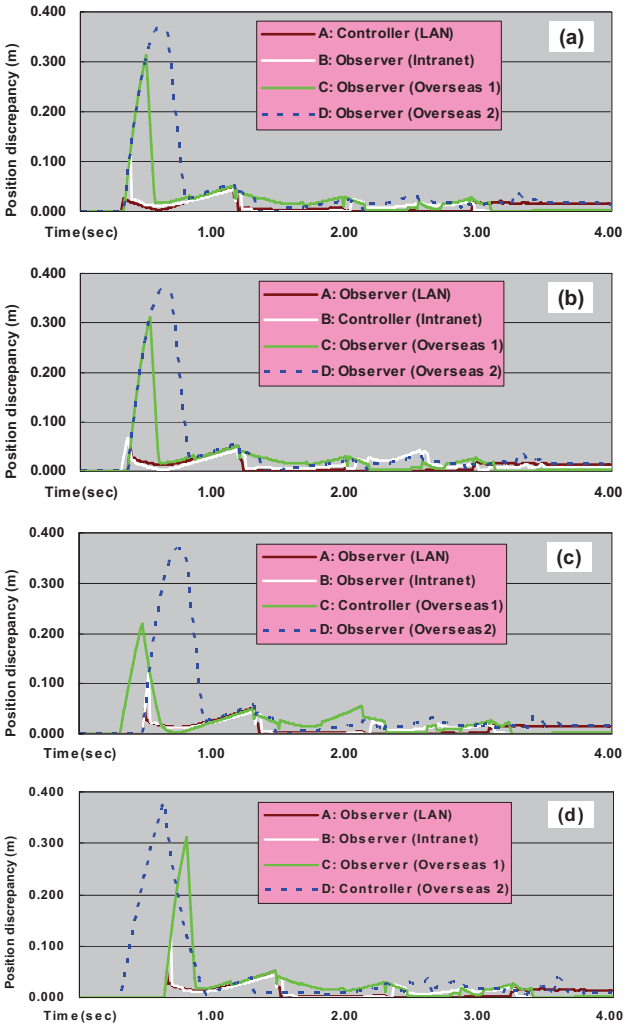

Fig. 7. Position discrepancies, during object collisions, of the four clients relative to the server when (a) client **A**, (b) client **B**, (c) client **C**, or (d) client **D**, is acting as the controller.

Since the controller throws out the selected object at around 0.3s, we can see from the four diagrams of Figure 7 that the position discrepancy between the controller and the server suddenly increases at 0.3s in all four diagrams. In Figure 7(a), as client A (the controller) has a rather low network latency with the server, it can quickly synchronize with the server and the discrepancy begins to drop until around 0.6s when the object reaches its highest point in the sky. Other observers will gradually synchronize with the server depending on their network latencies with the server. Once the server has received the message that the object has been thrown out from the controller's hand, the simulation program running in the server will take over the motion of the object. At 0.6s, the simulation program determines that the object should begin to fall down and accelerates as it falls due to gravity. Due to the error in the measured network latency as well as the fluctuation (or jittering) of the latency, which is further magnified by the acceleration of the object, the discrepancies between the four clients and the server gradually increase again until about 1.2s when the object hits the floor and rebounds.

Figures 7(b), 7(c) and 7(d) exhibit similar behavior, except that their observers' discrepancies with the server start at a later time after the controller has thrown the object out. This is due to the increase in network latency between the controller and the server.

7 Conclusion and Future Work

In this paper, we have proposed a synchronization method to support collaborative visualization. It considers how interaction with dynamic objects is perceived by application participants under the existence of network latency, and remedies the motion trajectory of the dynamic objects. It also handles the false positive and false negative collision detection problems. The new method is particularly well designed for handling content changes due to unpredictable user interventions or object collisions. Experimental results show that our method could effectively provide a good consistency control to support collaborative visualization.

Despite the merits of our proposed method, it does have some limitations. For example, it assumes using connection-oriented network protocols and message loss is not considered. We would like to address this as our future work. In addition, as haptic interfaces are becoming popular and it may widen the application of collaborative visualization by providing user with force feedback, we would also like to extend our work to handle haptic rendering as well.

Acknowledgements

We would like to thank the reviewers for their valuable comments and suggestions. The work described in this paper was partially supported by two CERG grants from the Research Grants Council of Hong Kong (Ref. Nos.: PolyU 5188/04E and CityU 1133/04E).

References

[1] V. Anupam, C. Baja, D. Schikore, and M. Schikore, "Distributed and Collaborative Visualization," *IEEE Computer*, **27**(7):37-43, Jul. 1994.

[2] P. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, **13**(2):185-221, 1981.

[3] Y. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," *Proc. of the Game Developers Conference*, 2001.

[4] S. Butner and M. Ghodoussi, "Transforming a Surgical Robot for Human Telesurgery," *IEEE Trans. on Robotics and Automation*, **19**(5):818-824, Oct. 2003.

[5] A. Chan, R. Lau, and B. Ng, "Notion Prediction for Caching and Prefetching in Mouse-Driven DVE Navigation," *ACM Trans. on Internet Technology*, **5**(1):70-91, Feb. 2005.

[6] O. Deusen et al., "The Elements of Nature: Interactive and Realistic Techniques," *ACM SIGGRAPH 2004 Course Note #31*, Aug. 2004.

[7] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet," *IEEE Networks Magazine*, **13**(4):6-15, Jul.-Aug. 1999.

[8] DIS Steering Committee, "IEEE Standard for Distributed Interactive Simulation - Application Protocols," *IEEE Standard 1278*, 1998.

[9] V. Jaswal, "CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the Cave Virtual Reality Theater," *Proc. of IEEE Visualization*, pp.301-308, Oct. 1997.

[10] G. Johnson and T. Elvins, "Introduction to Collaborative Visualization," *Proc. of ACM SIGGRAPH*, pp. 8-11, May 1998.

[11] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. on Computers*, **36**(8):933-940, Aug. 1987.
[12] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, **21**(7):558-565, 1978.
[13] L. Lamport and P. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *Journal of ACM*, **32**(1):52-78, Jan. 1985.
[14] F. Li, L. Li, and R. Lau, "Supporting Continuous Consistency in Multiplayer Online Games," *Proc. of ACM Multimedia*, pp. 388-391, Oct. 2004.
[15] C. Liao, M. Martonosi, and D. Clark, "Experience with an Adaptive Globally-Synchronizing Clock Algorithm," *Proc. of ACM SPAA*, pp.106-114, Jun. 1999.
[16] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz, "NPSNET: A Network Software Architecture For Large Scale Virtual Environments", *Presence: Teleoperators and Virtual Environments*, **3**(4):265-287, 1994.
[17] G. Mark, A. Kobsa, and V. Gonzalez, "Do Four Eyes See Better Than Two? Collaborative Versus Individual Discovery in Data Visualization Systems," *Proc. of IEEE Information Visualization*, pp. 249-255, Jul. 2002.
[18] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications," *IEEE Trans. on Multimedia*, **6**(1):47-57, 2004.
[19] D. Mills, "Internet Time Synchronization: the Network Time Protocol," *IEEE Trans. on Communications*, **39**(10):1482-1493, 1991.
[20] J. Ohlenburg, "Improving Collision Detection in Distributed Virtual Environments by Adaptive Collision Prediction Tracking," *Proc. of IEEE VR*, pp. 83-90, Mar. 2004.
[21] A. Pang and C. Wittenbrink, "Collaborative Visualization with Cspray," *IEEE Computer Graphics and Applications*, **17**(2):32-41, Mar.-Apr. 1997.
[22] S. Pettifer, J. Cook, J. Marsh, and A. West, "DEVA3: Architecture for a Large-Scale Distributed Virtual Reality System," *Proc. of ACM VRST*, pp. 33-40, Oct. 2000.
[23] T. Srikanth and S. Toueg, "Optimal Clock Synchronization," *Journal of ACM*, **34**(3):626-645, Jul. 1987.
[24] I. Vaghi, C. Greenhalgh, and S. Benford, "Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments," *Proc. of ACM VRST*, pp.42-49, Dec. 1999.
[25] J. Wijk, "Flow Visualization with Surface Particles," *IEEE Computer Graphics and Applications*, **13**(4):18-24, Jul. 1993.
[26] J. Wood, H. Wright and K. Brodlie, "Collaborative Visualization," *Proc. of IEEE Visualization*, pp. 253-259, Oct. 1997.
[27] M. Gerald-Yamasaki, "Cooperative Visualization of Computational Fluid Dynamics," *Proc. of Eurographics*, pp.497-508, Sept. 1993.